1. DispatcherServlet 将请求发送给SpringMVC控制器controller（需要有处理器映射handler mapping来帮助，因为有多个控制器）

之后发送给消息转换器（message converter）

之后把请求发送给（视图view，需要有视图解析器view resolver来帮助，因为有多个视图）

最后视图会将响应输出

1. Springmbvc的javaconfig

WebConfig 启动springmvc和加载控制器bean

@Configuration  
@EnableWebMvc //启动springmvc  
@ComponentScan("com.springjiemi.controller") //加载控制器bean  
public class WebConfig extends WebMvcConfigurerAdapter {  
}

WebAppInitializer Servlet会自动扫描这个类进行配置

//扩展AbstractAnnotationConfigDispatcherServletInitializer的任意类都会自动配置DispatcherServlet和Spring应用上下文  
 //spring应用上下文位于servlet应用上下文中  
 //容器会查找实现ServletContainerInitializer的类，用它来配置Servlet容器，它的spring默认实现为SpringServletContainerInitializer  
 //这个实现会查找实现WebApplicationInitializer的类，并将配置交给它来完成，它的基础实现为AbstractAnnotationConfigDispatcherServletInitializer  
 //所以我们扩展它就能配置Servlet容器  
public class WebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{  
 @Override  
 protected Class<?>[] getRootConfigClasses() {  
 return new Class<?>[]{RootConfig.class}; //指定组件bean（如数据层组件），此类需要带有@Configuration注解  
 }  
 @Override  
 protected Class<?>[] getServletConfigClasses() {  
 return new Class<?>[]{WebConfig.class}; //指定配置bean（如控制器，视图解析器，处理器映射），此类需要带有@Configuration注解  
 }  
 @Override  
 protected String[] getServletMappings() {  
 return new String[] {"/"}; //将DispatcherServlet映射到"/"  
 }  
}

//进行相关bean的配置

@Configuration  
@Import({ScanConfig.class, AspectConfig.class, DataSourceConfig.class})  
public class RootConfig {  
}

测试，编写控制器

@Controller  
@RequestMapping("/")  
public class TestController {  
 @RequestMapping(value = "jsontest.do")  
 @ResponseBody  
 public Map jsonTest() {  
 Map map = Maps.*newHashMap*();  
 map.put("test1", "xxxxx");  
 map.put("test2", "111111");  
 return map;  
 }  
}

[3.@RequestMapping注解 //](mailto:3.@RequestMapping注解%20//)指定访问路径

示例：

@RequestMapping(value = "jsontest.do")  
@ResponseBody  
public Map jsonTest() {  
 Map map = Maps.*newHashMap*();  
 map.put("test1", "xxxxx");  
 return map;  
}

参数：

Value 路径

Method 请求方法

[4.@RequestParam(“”)](mailto:4.@RequestParam() //指定请求参数名与装配参数的对应

示例：

@Controller  
@RequestMapping("/")  
public class TestController {  
 @RequestMapping(value = "jsontest.do")   
 @ResponseBody  
 public Map jsonTest(@RequestParam(value = "par", // 参数名defaultValue = "10" //默认值，如果该参数不存在的话

) int par1, @RequestParam("par2") String par3) {  
 Map map = Maps.*newHashMap*();  
 map.put("test1", "xxxxx" + par1 + par3);  
 return map;  
 }  
}

[5.@PathVariable](mailto:5.@PathVariable) 使用路径参数

示例：

@Controller  
@RequestMapping("/")  
public class TestController {  
 @RequestMapping("{requestId}")  
 @ResponseBody  
 public Map jsonTest(@PathVariable("requestId") int requestId) {  
 Map map = Maps.*newHashMap*();  
 map.put("test1", "xxxxx" + requestId);  
 return map;  
 }  
}

扩展：

1. 如果方法的参数名与路径中的占位符名相同，可以去掉@PathVariable的参数

示例：

@RequestMapping("{requestId}")  
 @ResponseBody  
 public Map jsonTest(@PathVariable int requestId) {  
 Map map = Maps.*newHashMap*();  
 map.put("test1", "xxxxx" + requestId);  
 return map;  
 }

1. 可以使用pojo对象作为参数，会自动装配

6.springmvc 配置multipart

示例：

Servlet配置文件中配置multipart详细信息

public class WebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{  
 //配置multipart具体细节  
 @Override  
 protected void customizeRegistration(ServletRegistration.Dynamic registration) {  
 registration.setMultipartConfig(new MultipartConfigElement("C:\\Users\\lenovo\\", 2097152, 4194304, 0));  
 }  
}

其中4个参数分别为:

Location //临时文件路径  
maxFileSize //上传文件的最大容量（字节为单位），默认无限  
maxRequestSize //整个multipart的最大容量，默认无限  
fileSizeThreshold //如果一个文件大小到了指定的最大容量，要不要存到临时路径中，默认为0，表示会。

springmvc配置文件中声明multipart bean

@Configuration  
@EnableWebMvc //启动springmvc  
@ComponentScan("com.springjiemi.controller") //加载控制器bean  
public class WebConfig extends WebMvcConfigurerAdapter {  
 //配置multipartResolver  
 @Bean  
 public MultipartResolver multipartResolver() {  
 return new StandardServletMultipartResolver();  
 }

}

[7.@ControllerAdvice](mailto:7.@ControllerAdvice) //为控制器添加通知

示例:

@ControllerAdvice  
public class AppWideExceptionHandler {  
 @ExceptionHandler(Exception.class)  
 public String xxxx() {  
 return "this is error";  
 }  
}

只要任何控制器抛出Exception异常，就会调用这个方法来处理异常

还有@InitBinder和@ModeAttribute注解也表明发出通知的条件

[8.@ExceptionHandler()](mailto:8.@ExceptionHandler()) //如果出现此异常会将请求转发给此控制器

示例：

@ExceptionHandler(Exception.class)  
public String xxxx() {  
 return "this is error";  
}

9. @ResponseStatus //如果控制器发生此异常，响应将会带有此状态码

示例：

@ResponseStatus(value = HttpStatus.*NOT\_FOUND*, reason = "Not found")  
public class NotFoundException extends RuntimeException{  
}

[10.@RestController](mailto:10.@RestController) 不再需要为每个方法都写上@esponseBody注解

示例：

@RestController  
@RequestMapping("/")  
public class TestController {  
 @RequestMapping("user/aaa.do")  
 public Map userdo() {  
 Map map = Maps.*newHashMap*();  
 map.put("user.do","user.aaa.do");  
 return map;  
 }

}

11. ResponseEntity 对象，返回带状态码元数据的对象

@RequestMapping("{requestId}")  
public ResponseEntity jsonTest(@PathVariable int requestId) {  
 Map map = Maps.*newHashMap*();  
 map.put("test1", "xxxxx" + requestId);  
 map.put("xhsfdo.do","xhsfdo.eee.do");  
 return new ResponseEntity(map, HttpStatus.NOT\_FOUND);  
}